

CONFIGURABLE VARIABLE LENGTH CODE WITH GENETIC ALGORITHMS

Ngai-Man Cheung and Yuji Itoh
E-mail: cheung@ti.com, yitoh@ti.com

Digital Consumer Electronics Solutions Technology Center, Texas Instruments Japan Ltd.

ABSTRACT

The variable length code (VLC) tables in MPEG-1/2/4 and H.26X are fixed and optimized for a limited range of bit-rates, and they cannot handle a variety of applications. The universal variable length code (UVLC) is a new scheme to encode syntax elements and has some configurable capabilities. It is being considered in the ITU-T H.26L. However, the configurable feature of UVLC has not been well explored. In this paper we describe a VLC scheme which uses configurations as parameters to adapt UVLC to different symbol distributions of different applications. We also propose a method to automatically determine the configuration parameters based on a genetic algorithm (GA). Experimental results show that our method can achieve very good coding efficiency while drastically simplifying the encoding and decoding process, and is applicable to a variety of applications.

1. INTRODUCTION

The variable length coding is a statistical coding technique that assigns symbols to codewords based on the occurrence frequency of the symbols. Symbols that occur more frequently are assigned short codewords while those that occur less frequently are assigned long codewords. Compression is achieved by the fact that overall the more frequent shorter codewords dominate. The variable length code (VLC) tables in MPEG-1/2/4 and H.26X are fixed and optimized for a limited range of bit-rates, and they cannot handle a variety of applications. For example, in MPEG-2 the VLC tables for DCT coefficients are designed for broadcast quality video applications, and they cannot readily handle low bit-rate applications. To further illustrate the problem, Figure 1 shows the probability distributions of the transform coefficient symbol at different bit-rates, using the H.26L video compression scheme. As shown in the figure, the distributions at different bit-rates are quite different, so using fixed VLC tables will not work well. Figure 2 compares the average bit per symbol with entropy, and reveals that there are large deviations at low quantizer scales (QP), indicating that the H.26L VLC scheme is not efficient at such bit-rates.

The *universal variable length code* (UVLC) was proposed in [1] as a new scheme to encode syntax elements and offers configurable advantage. Moreover, UVLC drastically simplifies the symbol encoding and decoding process. It is being considered in H.26L to code all the syntax elements [2]. However, the configurable feature of UVLC has not been well explored. In [3], we presented some works using the code length of the end of block (EOB) to adapt the UVLC to different bit-rate applications. In [4] the *dynamic symbol reordering* (DSR)

method was proposed to automatically re-configure the UVLC. The DSR uses the probability of each symbol to construct a mapping table that re-orders the assignments of symbols to codewords. The method is suitable for coding syntax elements which have only a few different symbols, e.g. the macroblock type in H.26L. It is not practical for syntax elements like transform coefficients or motion vectors, which consist of a lot of different symbols. Since the transform coefficients and motion vectors make up a significant portion of the total encoded bits, it is foremost important to be able to encode these syntax elements optimally in different types of applications.

In this paper we describe a configurable VLC scheme based on UVLC and a method to automatically determine the configuration. Section 2 of this paper describes the configurable VLC scheme, which uses *configurations* as parameters to adapt UVLC to different symbol distributions. Section 3 proposes a method to determine the configuration parameter based on a genetic algorithm (GA). The method can be applied to on-the-fly configuration of codewords during video encoding, or to off-line training of code tables. We also describe several techniques to reduce the required computation for applications which have only limited processing power. Section 4 presents the experimental results, and shows that the method can achieve very good coding efficiency while drastically simplifying the encoding and decoding process, and is widely applicable. Finally, we conclude the work in Section 5.

2. CONFIGURABLE VLC SCHEME

In this section we describe the configurable VLC scheme. It has a single, regular *codeword structure*, and can assume different probability distributions with different configurations as parameter. Figure 3 depicts the proposed scheme. The codeword structure is a UVLC, which divides the symbols into different *categories*, and assigns different *coarse code* (prefix) to each category. Within a category, the *additional code* (suffix) identifies an individual symbol. The coarse code follows the format "00...01". For the k th category, it has $k+1$ bits, k '0's followed by one '1' (Figure 3) [1].

The codeword structure can be instantiated to codewords with different probability distributions by using different additional codes for each category. The structure of the additional codes is defined by a configuration. The configuration is a one-dimensional array of L integers, $[r_k]_{k=0}^{L-1}$, where r_k is the code size of the additional code in the k th category, and L is the number of category. The final code is simply a concatenation of the coarse codes and additional codes. For example, Figure 4 shows the instantiated codewords with the configuration $[r_k]=[0,0,1,2,3,\dots]$.

As shown in Figure 3 our scheme has a very regular structure, and requires very simple encoding and decoding

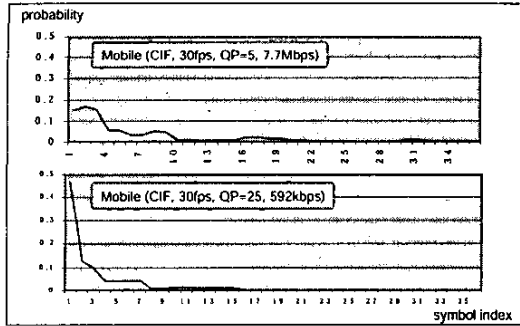


Figure 1. Probability distributions of the H.26L transform coefficient symbol at different bit-rates.

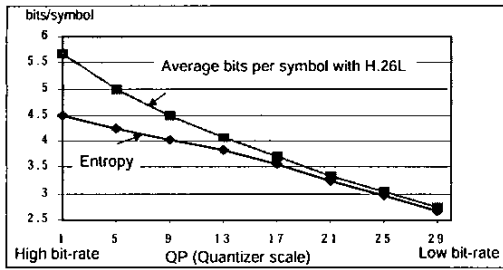


Figure 2. Comparing the average bits per symbol with entropy, using H.26L VLC scheme.

process compared with conventional VLC scheme in the existing coding standards. It can assume different probability distributions, and be instantiated to several popular coding schemes. For example, when configuration $[r_k] = [0, 1, 2, 3, 4, \dots, L-1]$, it resembles the *Elias Gamma* code, which is suitable for proportionally decreasing symbol distributions. When $[r_k] = [1, 1, 1, 1, \dots, 1]$, it resembles the *Golomb codes G(2)*. In general, for any integer $y \geq 0$, when $[r_k] = [y]$, then the scheme resembles Golomb codes $G(2^y)$. Figure 5 shows several examples.

3. DETERMINE THE CONFIGURATION

In this section we describe the configuration determination problem and propose a method to derive the optimal configuration. The optimal configuration has to meet two requirements. Firstly, it should *optimally partition* the symbols into different categories. Since the symbols within a category have the same code size, they should have similar occurrence probabilities, $2^{-code\ size}$, for optimal performance. In other words, the optimal configuration should partition the symbols such that symbols within a category have similar occurrence probabilities. As the symbol distributions at different bit-rates may be quite different, a fixed partition will not work optimally for all bit-rates. Let integer j , MIN_J , MAX_J denote the value, the minimum and the maximum of the symbols respectively, i.e., $MIN_J \leq j \leq MAX_J$, and t_0, t_1, \dots, t_{L-1} denote the boundary values of the categories. The k th category is $[t_k, t_{k+1}-1]$. The range of each category is 2^{r_k} . The configuration and the partition of symbols are related by the following equations

$$t_{k+1} = t_k + 2^{r_k} \quad (1)$$

$$t_0 = MIN_J \quad (2)$$

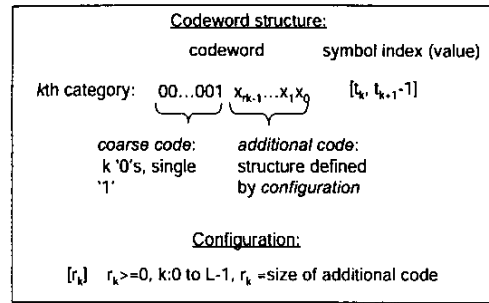


Figure 3. The configurable VLC scheme based on UVLC.

category	codeword	symbol index (value)	assumed probability
0	1	0	1/2
1	01	1	1/4
2	001 0	2	1/16
	001 1	3	
3	0001 00	4	1/64
	0001 01	5	
	0001 10	6	
	0001 11	7	
4	00001 000	8	1/256
...			

Figure 4. The instantiated codewords with configuration = $[0, 0, 1, 2, 3, \dots]$.

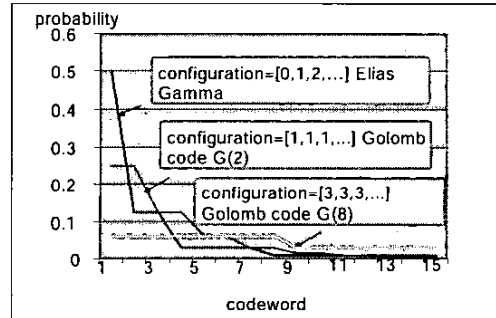


Figure 5. The configurable VLC scheme can assume different probability distributions.

Secondly, the optimal configuration should assign *optimal code size* to each category. If the average occurrence probability of the symbols in the k th category is p_k , then the code size of symbols in that category should be $-\log_2(p_k)$ for optimal performance. As the occurrence probabilities of symbols may be different for different bit-rates, we should adjust the code sizes accordingly, instead of having fixed code sizes for all bit-rates. Let cs_k be the code size of the k th category. The cs_k and configuration r_k are related by

$$cs_k = r_k + k + 1 \quad (3)$$

For optimal code size assignment, the configuration and the probability of symbols are related by

$$r_k = -(\log_2 p_k + k + 1) \quad (4)$$

The above two issues (i.e. partition and code size) are not independent, since how we assign the code sizes to the categories will affect the way we partition the symbols. This makes the problem of determining the optimal configuration

difficult. Let N_j be the number of occurrences of the symbol j . The total bits used to encode the symbols, B , is

$$B = \sum_{k=0}^{L-1} (cs_k \times \sum_{j=1}^{k+1} N_j) \quad (5)$$

An optimal configuration should minimize B .

Here we employ a genetic algorithm (GA) to determine the optimal configuration. GAs [5] are search procedure modeled on the mechanics of natural selection. They have been found effective in different search and combinatorial optimization problems, thanks to the implicit parallelism property (processing η^j possibilities in parallel with only η computation, where η is the size of population) and the optimality in *exploration* and *exploitation* of information (shown by an analogy to the *k-armed bandit* problem). Details can be found in [5].

GA maintains a population of candidate solutions encoded into bitstrings. In our scheme, we encode the configuration $[r_i]$ into bitstrings as shown in Figure 6. Table 1 lists the GA parameters, which are used in the experiments later. The values of the parameters are determined by several rules of thumb [5]. The maximum number of fitness function evaluation is 2500. The GA terminates when the total number of function evaluation equals to this number, or when the population converges. In most trials the optimal candidates can be found in several tens of function evaluations.

The GA evaluates the fitness of the candidate solutions using Equation (3) and (5), and this takes L multiplications and (MAX_J-MIN_J) additions. As we limit the number of fitness evaluation to 2500 per GA execution, the required numbers of operations are $2500 \cdot L$ multiplications, $2500 \cdot (MAX_J-MIN_J)$ additions. These are negligible for off-line applications. For on-line applications, suppose we try to determine the configuration on a frame-by-frame basis and assume an encoding speed of 30fps. If we code the runs of the transform coefficients of an 8×8 block, then $MIN_J=0$, $MAX_J=63$, and for $L=7$, the loading in determining the configuration is $0.525M$ multiplications/s and $4.725M$ additions/s. This is insignificant as an emerging digital signal processor (DSP) can deliver up to $800MMACS$ (million multiply-accumulate per second) at $0.05mW/MIPS$ (million instructions per second) [6].

There are some methods to reduce the loading for applications which have only limited processing power, and have only slight impact to the quality of results. Firstly, instead of using all the symbols for fitness calculation as in Equation (5), we use only the first few significant ones. For example, when coding the runs of transform coefficients, we may include only up to symbol value 32 (instead of 63) in the fitness function. This should be adequate in differentiating the candidate solutions in most cases.

There is another method to reduce required operations in particular for transform coefficients. We found that the distributions of runs and levels of transform coefficients are highly correlated among pictures of the same coding type (I-picture, P-picture or B-picture). So in many cases we can apply the configuration of the current picture to the succeeding pictures of the same coding type, and we do not have to perform the GA. Suppose picture S_0 has a symbol distribution DS_0 and configuration CS_0 . For each picture S_j succeeding S_0 of the same coding type, we apply the *chi-square test* to compare the distributions DS_0 and DS_j . If they are similar, we apply configuration CS_0 to picture S_j . If DS_0 and DS_j are different, we perform the GA on picture S_j to determine the optimal

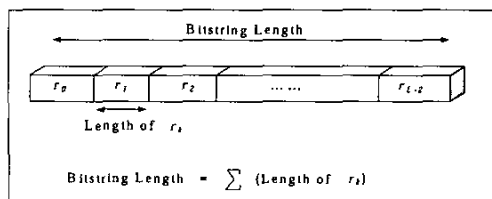


Figure 6. Bitstring encoding for determining configuration.

Table 1. GA parameters.

Maximum Number of Fitness Function Evaluations	2500
Population Size	50
Crossover Rate	0.6
Mutation Rate	0.001

configuration CS_j , and will apply CS_j to the succeeding pictures if DS_j and the distributions of the succeeding pictures are similar. The chi-square is

$$chi_square(D_{S_0}, D_{S_1}) = \sum_j \frac{(N_{j,S_0} - N_{j,S_1})^2}{N_{j,S_0} + N_{j,S_1}} \quad (6)$$

where N_{j,S_0} is the number of occurrences of the symbol value j in picture S_0 and N_{j,S_1} is that in picture S_1 . We use only the first few symbols instead of all of them in the chi-square test to save computations.

4. EXPERIMENTS

In this section we present the comparison results between the proposed coding scheme and the entropy coding scheme in H.26L TML-5 [2], and the VLC tables in MPEG-1/2.

We performed some simulations using our scheme to code the H.26L TCOEFF_Luma_SimpleScan (transform coefficient) syntax element. We did not change the mapping from symbol to codeword number, but only manipulated the codewords itself. Figure 7 shows the results for the 'foreman' sequence (QCIF 10fps) and the 'mobile' sequence (CIF 30fps). We used different configurations for different QP determined by GA, and the same configuration throughout the sequence. The number of categories L was set to 10. As shown in the figure, our scheme outperforms the H.26L entropy scheme in every QP, and by as much as 12.77% in some case. It is expected that we can further improve the results by changing the configuration dynamically as the sequence proceeds.

To understand how our method can outperform H.26L entropy scheme, we examined the probability distributions of our method, H.26L and distributions of symbols. The results are shown in Figure 8. As shown in Figure 8(a), our method can effectively match the symbol distribution especially at the region with small codewords. Also our method can readily adapt to a different symbol distribution as shown in Figure 8(b), which has a much lower bit-rate.

We also compared the proposed scheme to the VLC tables in MPEG-1/2. We measured the bits used in coding DCT coefficients. For our scheme we code the runs and levels separately as in [3]. Figure 9 shows the improvement. As shown in the figure our scheme outperforms the MPEG-1/2 VLC tables in every case when coding DCT coefficients, by 3% for MPEG-1 and 5% for MPEG-2 on average.

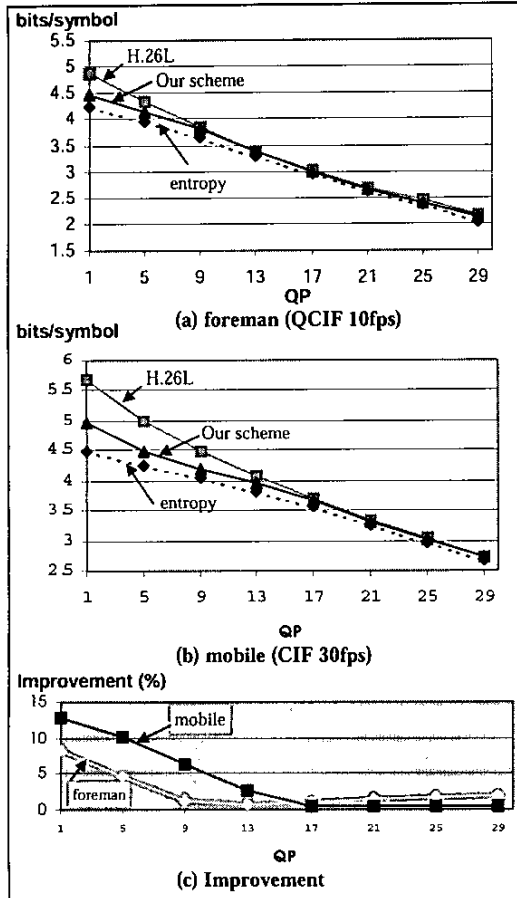


Figure 7. Comparing our scheme with the entropy scheme in H.26L when coding the transform coefficient.

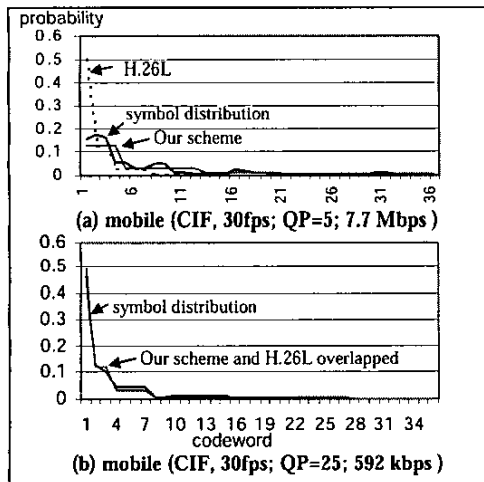


Figure 8. Probability distributions of symbols, H.26L VLC and our scheme, at different bit-rates.

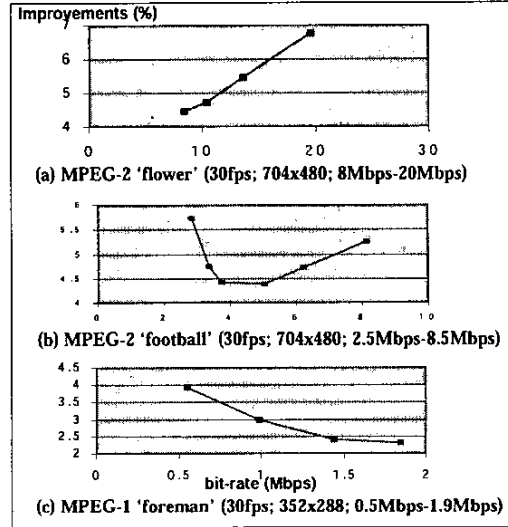


Figure 9. Comparing our scheme with the VLC tables in MPEG-1/2.

5. CONCLUSIONS

We have described a configurable VLC scheme based on GA. The GA is used to determine the configuration parameter, which adapts UVLC to different symbol distributions. The method is applicable to code the syntax elements with a lot of different symbols like the transform coefficients and motion vectors. We have presented the experimental results, and showed that the method consistently outperforms the entropy scheme in H.26L TML-5 and the VLC tables in MPEG-1/2 by bringing the bits consumption closer to the entropy over a large range of bit-rates. Our scheme can achieve such good coding efficiency while drastically simplifies the encoding and decoding process, and is applicable to a variety of applications. With these advantages it will be very useful for video coding.

6. REFERENCES

- [1] Yuji Itoh, "Bi-directional motion vector coding using universal VLC," Signal Processing: Image Communication, Vol. 14, pp. 541-557, May 1999.
- [2] G. Bjontegaard, "H.26L Test Model Long Term Number 5 (TML-5)," ITU-T Q.15/16, Doc. #Q15-K59, Oct. 2000.
- [3] Yuji Itoh and N.-M. Cheung, "Universal variable length code for DCT coding," in Proc. IEEE Int. Conf. Image Processing (ICIP), Vancouver, Canada, Sept. 10-13, 2000.
- [4] K.-Y. Yoo, B.-S. Choi and Y.-Y. Lee, "Improvements to the Telenor proposal for H.26L: Preliminary results on Dynamic Symbol Reordering (DSR) method for Universal VLC encoding/decoding," ITU-T Q.15/16, Doc. #Q15-H19, August, 1999.
- [5] Goldberg, D. E., Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Inc., 1989.
- [6] Texas Instruments TMS320C55x DSP Home Page at <http://www.ti.com/sc/docs/products/dsp/newcores/c55x.htm>.